Modul Pelatihan Teaching Factory Programming: Navigasi dan Lokalisasi dengan RTAB-Map

Program pelatihan ini dirancang untuk mahasiswa yang sudah memiliki pengetahuan dasar tentang ROS2. Fokus pelatihan adalah pada navigasi, lokalisasi, navigasi waypoint, dan pengendalian misi menggunakan state machine. Pelatihan ini menggunakan simulasi TurtleBot dan RTAB-Map.

1 Modul 1: Setup & Installasi

1.1 Konfigurasi Environment

- Install ROS 2 Humble lakukan installasi menggunakan **panduan instalasi Debian**
- Install TurtleBot3 packages menggunakan **panduan instalasi TurtleBot3**
- Install Gazebo dengan `sudo apt install ros-humble-desktop`
- Install RViz2 dengan `sudo apt install ros-humble-rviz2`
- Install RTAB-Map dengan `sudo apt install ros-humble-rtabmap-ros`
- Install Navigation2 dengan `sudo apt install ros-humble-navigation2 ros-humble-nav2-bringup`

1.2 Menjalankan Simulasi

Setelah semua paket terinstal, jalankan simulasi TurtleBot3 Waffle Pi di Gazebo dengan environment `house`:

```
export TURTLEBOT3_MODEL=waffle_pi
ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

1.3 Teleoperation

Buka terminal lain untuk menjalankan teleop:

```
ros2 run turtlebot3_teleop teleop_keyboard
```

1.4 RViz2

Buka RViz2 untuk melihat topik yang dipublish dan TF yang dimiliki robot:

```
rviz2
```

Atur Fixed Frame ke `odom`. Tambahkan display untuk `LaserScan`, dan `TF`. Sehingga akan muncul tampilan seperti berikut:

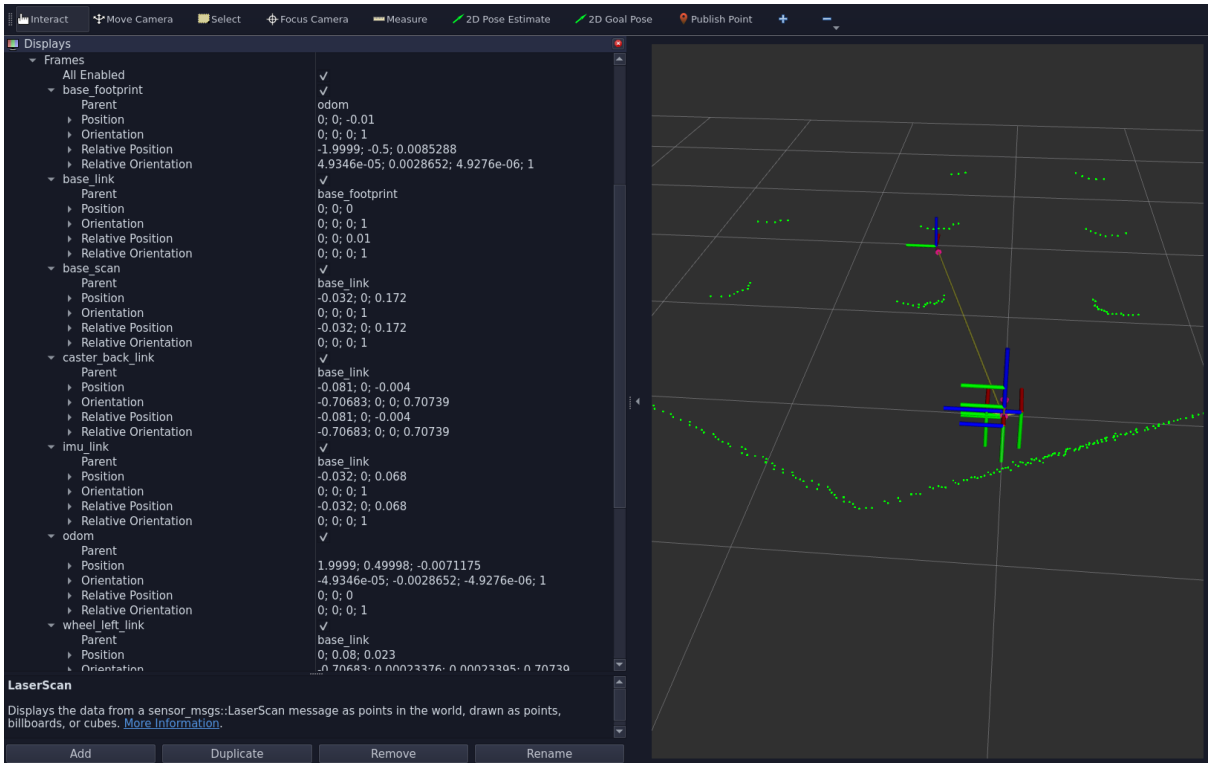

Figure 1: Contoh tampilan RViz2

Pada sidebar ditunjukkan TF yang dimiliki robot:

- `odom`: frame odometri, bergerak relatif terhadap `base_link`
- `base_footprint`: frame footprint robot (sejajar dengan ground plane 2D)
- `base_link`: frame utama robot (dapat mengalami translasi dan orientasi)
- `caster_back_link`: frame caster back (free wheel)
- `imu_link`: frame IMU
- `base_scan`: frame laser scan
- `wheel_left_link`: frame roda kiri
- `wheel_right_link`: frame roda kanan
- `camera_link`: frame dasar kamera (body), menunjukkan posisi fisik kamera pada robot. Orientasi mengikuti konvensi ROS, yaitu x ke depan, y ke kiri, dan z ke atas.
- `camera_rgb_frame`: frame kamera RGB (sensor), digunakan sebagai acuan untuk data gambar. Orientasinya sama dengan `camera_link`.
- `camera_rgb_optical_frame`: frame kamera dengan konvensi optik. Orientasi mengikuti standar OpenCV, yaitu z ke depan, x ke kanan, dan y ke bawah.

Semua TF tersebut terhubung dalam sebuah pohon (tree) yang dapat dilihat pada tab `TF` di RViz2. atau dengan perintah:

```
ros2 run tf2_tools view_frames.py
```

Perintah ini akan menghasilkan file `frames.pdf` yang menunjukkan struktur pohon TF sebagai berikut:

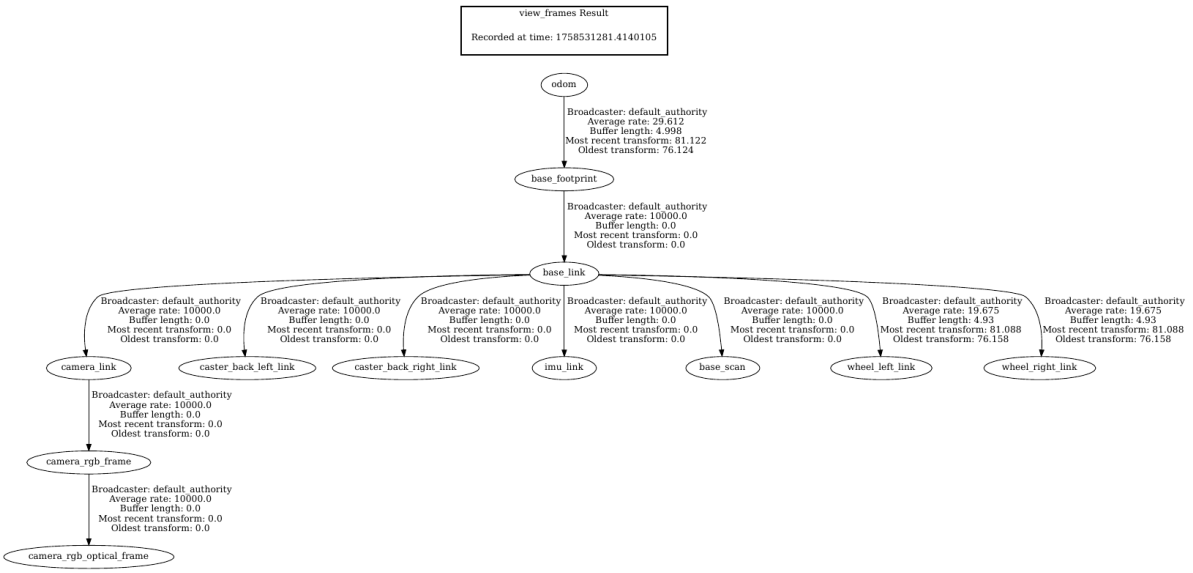

Figure 2: Contoh TF Tree

2 Modul 2: Mapping dengan RTAB-Map

2.1 SLAM dan RTAB-Map

SLAM (Simultaneous Localization and Mapping) adalah proses di mana robot membangun peta lingkungan sambil menentukan posisinya di dalam peta tersebut. Pada dasarnya, SLAM dibagi menjadi dua yaitu map based SLAM dan graph based SLAM. Perbedaan paling mendasar adalah pada bagian lokalisasinya. Pada map based SLAM, lokalisasi dilakukan dengan melakukan registrasi data sensor ke map yang telah dibangun sebelumnya. Sedangkan pada graph based SLAM, lokalisasi dilakukan dengan registrasi sensor ke setiap graph yang dibangun sebelumnya.

RTAB-Map (Real-Time Appearance-Based Mapping) adalah sebuah algoritma graph based SLAM yang menggunakan data visual (gambar) dan data jarak (misalnya dari LIDAR atau depth camera) untuk membangun peta 3D dari lingkungan sekitar. RTAB-Map dapat digunakan untuk berbagai aplikasi robotika, termasuk navigasi, inspeksi, dan pemetaan lingkungan. Untuk memperjelas bagaimana cara kerja Graph Based SLAM, silahkan simak video berikut: **Graph Based SLAM Explanation**.

2.2 RTAB-Map

Kunci dari Graph Based SLAM adalah deteksi loop closure. Di RTAB-Map, deteksi loop closure menggunakan kamera. Algoritma dasar yang perlu dipelajari untuk mencari loop closure menggunakan kamera adalah Image Registration. Image Registration adalah proses mencocokkan dua gambar yang diambil dari sudut pandang yang berbeda untuk menemukan transformasi geometris antara keduanya. Contoh contoh algoritma yang sering dipakai adalah GFTT, FAST, BRIEF, ORB, SIFT, dan lain lain.
RTAB-Map bukanlah Ready to Use SLAM, perlu dilakukan penyesuaian pada robot yang akan digunakan!

2.3 Penyesuaian RTAB-Map

2.3.1 Penyesuaian Sensor

Di RTAB-Map, bisa menggunakan beberapa sensor, cara setting nya lewat parameter:

- **subscribe_depth**: Untuk kamera depth seperti realsense
- **subscribe_scan**: Untuk LIDAR 2d
- **subscribe_scan_cloud**: Untuk LIDAR 3d

- **subscribe_rgbd**: Untuk kamera RGBD seperti Kinect
- **subscribe_stereo**: Untuk kamera stereo seperti Zed
- **subscribe_odom**: Untuk odometri hardware atau sumber odometry lainnya

Jika sistem robot tidak memiliki source odometry, odometry bisa dihitung dari visual odometry atau icp odometry bawaan RTAB-Map. Untuk menggunakannya bisa menajalankan node lain yaitu `rgbd_odometry` atau *icp_odometry*.

2.3.2 Penyesuaian Loop Closure Detection

Ada beberapa hal penting yang perlu diperhatikan dalam penyesuaian loop closure detection:

- **Kp/DetectorStrategy**: Memilih algoritma image matching yang sesuai dengan kondisi lingkungan dan jenis kamera yang digunakan.
- **Kp/MaxFeatures**: Berapa banyak maksimal fitur (words) yang bisa disimpan dalam database. Perhatikan juga setiap algoritma akan menghasilkan jumlah fitur yang berbeda beda.
- **Vis/MinInliers**: Hasil dari scan matching juga berupa inliers yang telah diperoleh dari RANSAC. Semakin besar nilai inliers, semakin baik hasil scan matching. Namun, jika terlalu besar, bisa jadi tidak ada loop closure yang terdeteksi.

2.3.3 Penyesuaian Graph Optimization

Graph optimization adalah proses memperbaiki posisi node pada graph berdasarkan loop closure yang terdeteksi. Ada beberapa parameter penting yang perlu diperhatikan:

- **Reg/Strategy**: Memilih mau menggunakan sensor apa untuk menghitung constraint per node pada graphnya. Bisa menggunakan icp atau visual atau gabungan keduanya. ICP dari LIDAR sedangkan visual dari kamera.
- **Optimizer/Strategy**: Memilih algoritma optimizer yang diinginkan (default GTSAM).
- **Optimizer/Iterations**: Semakin besar maka semakin lama iterasi optimasinya, hasilnya belum tentu lebih baik. Gunakan default saja jika tidak ada masalah.
- **RGBD/OptimizeMaxError**: Gerbang terakhir untuk optimasi, Jika hasil error nya terlalu besar maka loop closure akan di-reject dan graph tidak akan dioptimasi.
- **RGBD/OptimizeFromGraphEnd**: Ketika loop closure diterima, ada dua pendekatan optimasi, optimasi seluruh graph atau hanya dari node terakhir. Jika optimiasi dari seluruh graph, pasti akan membuat map atau estimasi pose robot loncat.

2.3.4 Parameter Penting Lainnya

- **Mem/IncrementalMemory**: Dipastikan false, agar database dalam keadaan frozen saat pertama kali dihidupkan. Intinya, keadaan default adalah keadaan lokalisasi, bukan mapping. Jika ingin mapping, harus diaktifkan dengan service.
- **Mem/STMSize**: Berapa banyak node terakhir yang akan disimpan di Short Term Memory (STM). Node-node ini akan dibandingkan dengan node baru untuk mencari loop closure. Jika robot bergerak cepat, maka STM size harus lebih besar.
- **Mem/NotLinkedNodesKept**: Berapa banyak node yang tidak terhubung dengan loop closure yang akan disimpan di memori. Setting ini ke True agar databse tidak crash.
- **RGBD/NeighborLinkRefining**: Jika TF sensor sudah dipastikan bagus, maka parameter ini bisa diset True. Jika kurang percaya dengan hasil sensor, dan ingin lebih percaya dengan odometry, maka bisa diset False.

2.3.5 Grid Map

Grid hasil dari RTAB-Map berupa occupancy grid. Hasil dari occupancy grid ini tidak penting untuk lokalisasi karena RTAB-Map merupakan graph based SLAM. Biasanya, grid map ini digunakan untuk navigasi yang nantinya berupa global costmap. Ada beberapa parameter penting yang perlu diperhatikan:

- **Grid/Sensor**: Jenis sensor yang digunakan untuk membuat grid. Bisa menggunakan laser scan, depth image, atau point cloud. Pilih sesuai dengan sensor yang dimiliki robot.
- **Grid/RangeMax**: Jarak maksimal dari sensor yang akan dimasukkan ke dalam grid. Jika jarak sensor lebih dari nilai ini, maka data tersebut akan diabaikan.
- **Grid/CellSize**: Ukuran sel grid. Semakin kecil ukuran sel, semakin detail peta yang dihasilkan, namun juga semakin besar ukuran peta dan semakin lama waktu pemrosesan.

3 Module 3: Navigation with Nav2

3.1 planner_server

Planner server bertanggung jawab untuk merencanakan jalur dari posisi awal ke tujuan. Planner server menggunakan algoritma perencanaan jalur seperti A* atau Dijkstra untuk menemukan jalur terpendek yang menghindari obstacle. Planner server menerima permintaan perencanaan jalur dari client (misalnya, `nav2_bt_navigator`) dan mengembalikan jalur yang direncanakan dalam bentuk serangkaian pose (waypoints) yang harus diikuti oleh robot.

Parameter Penting planner_server

Beberapa parameter penting pada `planner_server` yang perlu diperhatikan:

- **use_sim_time**: Pilih `true` untuk simulasi, `false` untuk real time.
- **expected_update_rate**: Frekuensi update perencanaan jalur (dalam Hz).
- **planner_plugin**: Plugin planner yang digunakan untuk perencanaan jalur.

Plugin Planner pada planner_server

Plugin	Deskripsi dan Penggunaan
NavFn Planner	Algoritma grid-based (Dijkstra/A*) untuk jalur terpendek. Cocok untuk omni, mecanum, dan differential drive robot.
Smac 2D Planner	Grid-based A* dengan cost-awareness. Implementasi modern, cocok untuk omni, mecanum, dan differential drive robot.
Smac Hybrid-A* Planner	Mempertimbangkan heading robot pada tiap node, menghasilkan path yang lebih realistis secara kinematik. Cocok untuk car-like robot (non-holonomic) dan ackerman steering.
Smac State Lattice Planner	Planner berbasis State Lattice dengan motion primitives precomputed (offline). Cocok untuk car-like robot (non-holonomic) dan ackerman steering.
Theta* Planner	Algoritma Theta*, variasi A* yang memungkinkan any-angle paths (tidak terbatas pada grid horizontal/vertikal). Cocok untuk omni, mecanum, dan differential drive robot.

Table 1: Plugin Planner pada planner_server Nav2

3.2 controller_server

`Controller_Server` bertanggung jawab untuk mengendalikan gerakan robot agar mengikuti jalur yang telah direncanakan oleh `planner_server`. `controller_server` menerima jalur (path) dari `planner_server` dan menghasilkan perintah kecepatan (velocity commands) yang akan dikirim ke robot untuk mengikuti jalur tersebut.

Parameter Penting controller_server

Beberapa parameter penting pada `controller_server` yang perlu diperhatikan:

- **use_sim_time**: Pilih `true` untuk simulasi, `false` untuk real time.
- **controller_frequency**: Frekuensi kontrol (dalam Hz).
- **odom_topic**: Topik odometri yang digunakan untuk mendapatkan informasi posisi dan kecepatan robot.
- **progress_checker_plugin**: Plugin yang digunakan untuk memeriksa kemajuan robot menuju tujuan.
- **goal_checker_plugins**: Plugin yang digunakan untuk memeriksa apakah robot telah mencapai tujuan.
- **controller_plugins**: Plugin yang digunakan untuk mengendalikan gerakan robot mengikuti jalur.

Plugin Progress Checker pada controller_server

Plugin	**Deskripsi dan Penggunaan**
SimpleProgressChecker	Cocok untuk sebagian besar robot dan lingkungan. Memantau apakah robot membuat kemajuan yang cukup menuju tujuan berdasarkan jarak yang ditempuh.
PoseProgressChecker	Cocok untuk robot yang memerlukan kontrol orientasi yang presisi. Memantau kemajuan posisi dan orientasi, memastikan robot tidak hanya bergerak menuju tujuan tetapi juga berorientasi dengan benar.

Table 2: Plugin Progress Checker pada controller_server Nav2

Plugin Goal Checker pada controller_server

Plugin	**Deskripsi dan Penggunaan**
SimpleGoalChecker	Cocok untuk sebagian besar robot dan lingkungan. Memeriksa apakah robot berada dalam toleransi jarak dan orientasi tertentu dari tujuan.
StoppedGoalChecker	Cocok untuk robot yang perlu berhenti sepenuhnya di tujuan. Selain toleransi jarak dan orientasi, memeriksa apakah kecepatan linier dan angular robot berada di bawah ambang batas tertentu.
PositionGoalChecker	Cocok untuk robot yang hanya perlu mencapai posisi tertentu tanpa persyaratan orientasi yang ketat. Memeriksa apakah robot berada dalam toleransi jarak tertentu dari posisi tujuan, mengabaikan orientasi.

Table 3: Plugin Goal Checker pada controller_server Nav2

Plugin Controller pada controller_server

Plugin	Deskripsi dan Penggunaan
DWBLocalPlanner	Cocok untuk robot differential drive di lingkungan dinamis. Menggunakan pendekatan dynamic window untuk menilai trajektori berdasarkan kinematika dan costmap lokal.
Regulated Pure Pursuit Controller	Cocok untuk robot yang memerlukan pengikutan jalur yang halus dan akurat. Menggunakan titik lookahead pada jalur untuk menentukan perintah kemudi, dengan regulasi kecepatan linier berdasarkan kelengkungan jalur dan kedekatan rintangan.

Table 4: Plugin Controller pada controller_server Nav2

3.3 recoveries_server

Recoveries server bertanggung jawab untuk menjalankan tindakan pemulihan (recovery behaviors) ketika robot gagal dalam perencanaan atau navigasi. Misalnya, jika robot terjebak atau tidak dapat menemukan jalur, maka recovery behaviors akan membantu robot keluar dari kondisi tersebut.

Parameter Penting recoveries_server

- **use_sim_time**: Pilih `true` untuk simulasi, `false` untuk real time.
- **costmap_topic**: Topik costmap yang digunakan untuk mendeteksi obstacle.
- **recovery_plugins**: Daftar plugin recovery yang dapat digunakan.

Plugin Recovery pada recoveries_server

Plugin	Deskripsi dan Penggunaan
Spin Recovery	Memutar robot di tempat untuk mencari jalur bebas obstacle.
Backup Recovery	Memundurkan robot sejauh jarak tertentu agar keluar dari deadlock.
Wait Recovery	Robot berhenti sejenak, berguna jika obstacle yang menghalangi bersifat sementara (misalnya orang lewat).
Clear Costmap	Menghapus data obstacle di costmap lokal/global agar perencanaan jalur bisa diperbarui dengan kondisi terkini.

Table 5: Plugin Recovery pada recoveries_server Nav2

3.4 bt_navigator

Behavior Tree (BT) Navigator adalah komponen utama yang mengatur alur navigasi menggunakan *behavior tree*. BT memungkinkan fleksibilitas dan modularitas dalam mendefinisikan urutan aksi, percabangan, serta kondisi navigasi.

3.5 global_costmap

Global costmap bertugas membuat representasi peta statis dan dinamis yang digunakan oleh global planner untuk merencanakan jalur dari posisi awal ke tujuan. Global costmap biasanya mencakup area besar, sehingga mampu melihat keseluruhan lingkungan.

Parameter Penting global_costmap

- **global_frame**: Frame referensi global, biasanya `map`.
- **robot_base_frame**: Frame dasar robot, biasanya `base_link`.
- **update_frequency**: Frekuensi update data costmap (Hz).
- **resolution**: Resolusi grid (m/cell). Semakin kecil nilainya, semakin detail tetapi lebih berat komputasinya.
- **robot_radius**: Radius robot (m) yang digunakan untuk inflasi obstacle.
- **plugins**: Layer yang digunakan, biasanya:
  - Static Layer: Mengambil data peta statis (dari SLAM/Map server).
  - Obstacle Layer: Menggunakan sensor (Lidar/sonar).
  - Inflation Layer: Memberikan jarak aman dari obstacle.

Batasan Tuning global_costmap

- **resolution**: Jangan terlalu kecil (< 0.02 m/cell) karena memori dan CPU bisa overload.
- **width/height**: Sesuaikan dengan ukuran map (misalnya 50x50 m). Terlalu besar akan membuat costmap berat.
- **robot_radius vs. inflation_radius**: Jika `inflation_radius < robot_radius`, robot bisa menabrak obstacle. Jika terlalu besar ($> 3\times$ ukuran robot), jalur bisa tidak ditemukan.

3.6 local_costmap

Local costmap digunakan oleh local planner (controller server) untuk menghindari obstacle secara dinamis. Local costmap biasanya rolling window yang mengikuti pergerakan robot, dengan area yang relatif kecil.

Parameter Penting local_costmap

- **global_frame**: Biasanya `odom`, karena hanya lokal dan tidak perlu akurat ke peta global.
- **robot_base_frame**: Frame dasar robot, biasanya `base_link`.
- **rolling_window**: Harus `true`, agar peta mengikuti robot.
- **width/height**: Lebar dan tinggi window (misalnya 6x6 m).
- **resolution**: Grid size (misalnya 0.05 m).
- **plugins**:
  - Obstacle Layer: Membaca sensor (lidar, sonar, bumper).
  - Inflation Layer: Menjaga jarak aman.

Batasan Tuning local_costmap

- **width/height**: Jangan terlalu besar (> 10 m), karena costmap lokal menjadi lambat diproses.
- **robot_radius**: Sesuaikan dengan dimensi robot sebenarnya. Jika terlalu besar, robot akan menganggap semua jalur sempit tidak bisa dilewati.
- **inflation_radius**: Umumnya antara $1.2\times$ sampai $2\times$ `robot_radius`. Jika terlalu kecil, robot bisa terlalu dekat dengan obstacle; jika terlalu besar, robot bisa gagal menemukan jalur.
- **update_frequency**: Disarankan 5–10 Hz. Terlalu tinggi bisa membebani CPU, terlalu rendah membuat respon robot lambat.